

# Maximizing CMP Throughput with Mediocre Cores

John D. Davis, James Laudon<sup>†</sup>, Kunle Olukotun

Stanford University  
{johnd, kunle}@stanford.edu

<sup>†</sup>Sun Microsystems, Inc.  
James.Laudon@Sun.COM

## Abstract

*In this paper we compare the performance of area equivalent small, medium, and large-scale multithreaded chip multiprocessors (CMTs) using throughput-oriented applications. We use area models based on SPARC processors incorporating these architectural features. We examine CMTs with in-order scalar processor cores, 2-way or 4-way in-order superscalar cores, private primary instruction and data caches, and a shared secondary cache. We explore a large design space, ranging from processor-intensive to cache-intensive CMTs. We use SPEC JBB2000, TPC-C, TPC-W, and XML Test to demonstrate that the scalar simple-core CMTs do a better job of addressing the problems of low instruction-level parallelism and high cache miss rates that dominate web-service middleware and online transaction processing applications. For the best overall CMT performance, smaller cores with lower performance, so called “mediocre” cores, maximize the total number of CMT cores and outperform CMTs built from larger, higher performance cores.*

## 1. Introduction

The research community has been predicting the genesis of chip multiprocessors (CMPs) for some time [21]. Starting around the 130 nm silicon process generation, it was possible to put multiple superscalar processor cores on a single chip [10][11]. These complex CMPs are becoming the ubiquitous architecture for commercial servers targeting throughput-oriented applications. However, using wide-issue superscalar processor cores in CMTs has its drawbacks. These complex cores are focused on instruction level parallelism (ILP) extraction and high clock frequency, yielding devices able to execute many billions of instructions per second under ideal conditions. Unfortunately, this massive instruction processing capability is throttled by the large latency gap between the memory subsystem and the processor, and for many larger commercial applications, only a

tiny fraction of the peak performance can be achieved [4]. Now that there is enough die real estate to produce large-scale CMTs, we believe it is necessary to re-evaluate the underlying microarchitecture to determine the optimal processor building block for targeting commercial workloads.

Most important commercial server applications, such as e-commerce, online transaction processing (OLTP), decision support systems (DSS), and enterprise resource planning (ERP) are heavily threaded, and even for non-threaded applications, there is a trend towards aggregating those applications to run on a common server pool, as exemplified by grid computing or “computation on demand.” For these workloads, the individual thread latency is less important than the aggregate thread throughput. We increase aggregate thread throughput by using multithreaded processor to hide events that normally stall the processor, such as cache misses, thereby increasing their utilization. By using multithreading architectures that employ no-overhead thread switching [16][29], processors can come close to their peak computation rate even in the presence of low ILP and high cache miss rates, at a relatively modest hardware cost [28][29]. Within the multithreaded architecture space, the main processor trade-off involves thread latency versus thread count. The CMT can either employ a smaller number of more-powerful processors emphasizing individual thread latency, but sacrificing aggregate thread throughput, or employ a larger number of less-powerful processors emphasizing aggregate thread throughput, but degrading individual thread latency. The POWER 5™ [6] and Niagara [13] processors best illustrate this thread performance spectrum, respectively. The less-powerful processor approach has an intuitive appeal, as a simple, scalar processor requires much less area than a superscalar processor while still providing similar sustainable performance achievable on an n-wide superscalar processor running commercial benchmarks. In addition, using simple, scalar processor cores reduces the design complexity and bug rates.

To investigate the CMT design space, we use Sun Microsystems’ processor databases to generate and correlate the area models for both scalar and in-order

superscalar processors employing fine-grain multithreading. We use a variety of industry guidelines to reduce the number of simulations in the design space. Even with these guidelines, approximately 13,000 configurations exist for in-order scalar CMTs for each benchmark and process technology. Results from a perfect L2 cache simulator configuration enabled further pruning of the scalar CMT design space. Superscalar processor configurations were explored to enable comparisons to previous studies that investigate CMPs [5][9]. Unlike these previous studies, our superscalar processors include multithreading, which we show is crucial to achieve high throughput.

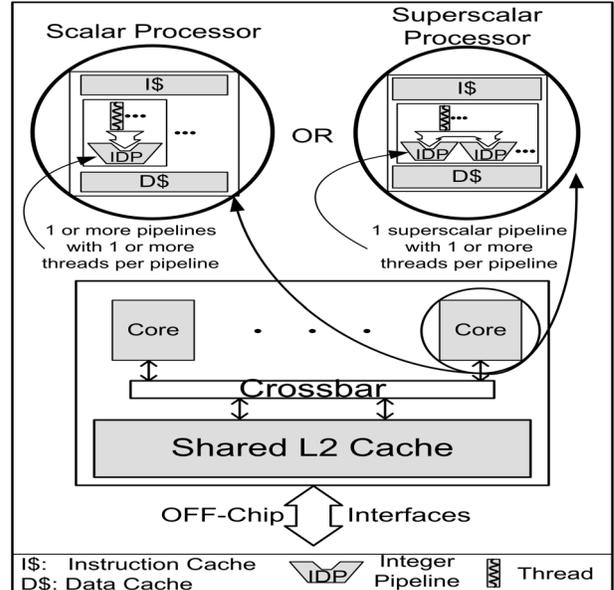
The design trade-offs and complexity for CMTs composed of a large number of simple processors are very different from that of a CMP consisting of a small number of superscalar processor cores. In this paper, we use SPEC JBB [30], TPC-C, TPC-W [31] and XML Test [32], a Java middleware, OLTP, transactional web, and XML parsing web benchmarks, respectively. For these applications, total throughput, rather than single-thread performance, is the main metric of interest. We measure total throughput using aggregate instructions per cycle (AIPC), which we find to be directly related to transactions per second for our highly tuned versions of the benchmarks. For all benchmarks, AIPC is maximized for a range of scalar CMT configurations employing small primary caches with roughly 25-40% of the CMT area devoted to shared secondary cache area. We observed consistent trends across technologies that enable us to extrapolate our results from small-scale and medium-scale CMTs to large-scale CMTs.

We discuss the CMT design space and describe our area model based on various processor core components and cache designs used to determine the allowable CMT configurations in Section 2. Section 3 elaborates on our high performance multi-configuration simulation environment. Section 4 presents the detailed results of our simulations. Section 5 discusses related work and we conclude in Section 6.

## 2. The CMT design space

We evaluated CMTs built from processor cores implementing the SPARC ISA. By exploring several of Sun Microsystem’s UltraSPARC chip design databases, we determined the area impact of the architectural components that are modified to enable fine-grain multithreading. From this, we derived a thread-scalable fine-grained multithreaded processor core area model, which correlates well with actual and projected UltraSPARC processor areas from 130 nm to 45 nm silicon process generations. We present simulated

results for small-scale, medium-scale, and (limited) large-scale CMTs, where small, medium, large classify CMT configurations that correspond to reticle-limited dies (400 mm<sup>2</sup>) for 130 nm, 90 nm, and 65 nm silicon process technologies, respectively



**Figure 1: A high-level functional diagram of the CMT design space. The gray components are varied and described in Table 1.**

**Table 1: CMT design space parameters.**

Feature	Description
CPU	In-order scalar or superscalar
Issue Width	scalar, 2-way and 4-way superscalar
Pipeline Depth	8 stages
Integer Datapath Pipelines	1-4 IDPs or Integer ALUs
L1 D & I Cache	8KB-128KB, 16 (D) & 32 (I) Byte lines
L1 D & I Cache Set Assoc.	Direct-mapped, 2-, 4-, or 8-way
L1 D & I Cache Policies	write through, LRU-based replacement
Clock Frequency	1/3 -1/2 Maximum ITRS clock frequency [23]
Multithreading	1-32 threads/core
L2 Cache	1MB - 8MB, 128 Byte lines, banked (8 or 16), coherent, inclusive, shared, unified, critical word first, 25 cycle hit time (unloaded)
Main Memory	Fully Buffered DIMMs with 4/8/16 dual channels, 135 cycle latency (unloaded)

Figure 1 illustrates and Table 1 describes the variety of high-level CMT configurations; all the gray components are varied in this study. The processor cores can utilize either in-order scalar or superscalar integer datapaths (IDPs). We vary the number of IDPs within each core and the number of threads per IDP. In our scalar processor design, threads are statically assigned to an IDP, as this avoids the superlinear area impact of being able to issue instructions from any of the threads on a core to any of the IDPs. All cache sizes and set associativities (SA) can vary. Instruction

caches and data caches are always identical in size or differ by a factor of 2X, but no more. The primary caches range from 8 KB to 128 KB with SA ranging from direct mapped to 8-way. Small instruction buffers for each thread decouple the front-end of each IDP from the shared primary instruction cache. The memory and cache subsystems are fully modeled with queuing delaying and occupancy. The actual RAS/CAS cycles for the DRAM accesses are modeled along with all the various buffers and queues. The number of processor cores and sizes of the caches are determined by the area model for a given silicon process technology, keeping die size constant across all possible configurations.

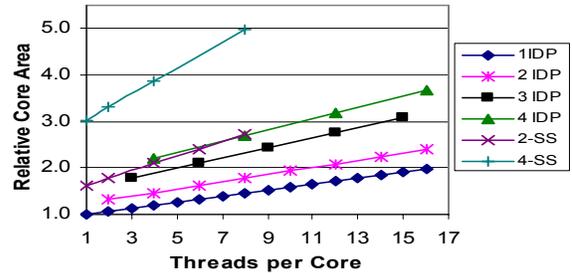
## 2.1. The CMT area model

Historically, server microprocessors have pushed the manufacturing envelope close to the reticle limit, around 400 mm<sup>2</sup>. Hence, we fixed the die size to be 400 mm<sup>2</sup> across the technology generations and allocate 75% of the total die area to the CMT area, processor cores and secondary cache, with the remaining 25% devoted to the other system-on-a-chip (SOC) components: memory controllers, I/O, clocking, etc. We devote 15% of the CMT area to the processor core interconnect and related components and the remaining 85% of the CMT area (60% of the total area) is devoted to the processor cores and secondary cache. The number of processor cores and the size of the secondary cache are determined by allocating between 25% to 75% to one and the remainder of the area to the other to cover a broad range of CMT configurations, from processor intensive to on-chip memory intensive designs. We also account for spacing and routing between the (sub)components; an additional 10-20%, depending on the component, of die area is allocated for this purpose at various levels of the area model. Thus, our area model produces realistic CMT configurations that have been validated against Sun Microsystem’s processor designs.

## 2.2. Processor core & cache area

From our estimates, fine-grain multithreading directly impacts the area of processor core components in a linear manner for a small number of threads, but the degree to which these components are affected varies greatly. We estimate a 5-6% area increase when integrating two active threads into a simple, in-order scalar or superscalar processor. This area increase is similar to the area increase due to simultaneous multithreading reported by Intel and IBM [19][6]. Figure 2 illustrates the linear increase in processor core

area predicted by our model, for a maximum of 16 hardware threads per processor core. The number of threads per core is shown on the x-axis and the y-axis quantifies the relative area increase of the core configuration when adding multithreading and additional IDPs. The increased complexity and resulting non-linear area increase for large number of threads per IDP is not modeled. This non-linearity is realized much sooner with superscalar cores, preventing our area model from accurately predicting superscalar cores with more than 8 threads.



**Figure 2: Core area model relative to a scalar single-thread single IDP core for scalar (X IDP) and superscalar (Y SS), where X is the number of IDP sharing private primary caches and Y is the instruction issue width.**

We initially used CACTI 3.2 [25] to estimate cache area and power, but found some inaccuracies and limitations for the sub-micron silicon process generations that we were interested in modeling [33]. We use conservative area estimates based on Sun Microsystem’s designs for cache memory cells with an area efficiency of 50% for all processor configurations. Based on access time limitations, we constrained the primary cache size with respect to the other processor components to be no more than 50% of the total processor core area. This constraint favors larger cores, as our simulations do not assume multiple-cycle access for the larger primary caches, and so are optimistic in their performance benefits. While this might seem to give an advantage to large thread or large IDP configurations, we show in Section 4.3, all but two of the best performing configurations use small primary caches that were available to all possible thread/IDP combinations, with the two outliers using caches available to all combinations except the lower-thread count, single IDP configurations.

## 3. Simulation environment

While our simulation study encounters the standard problems of simulating nonexistent systems and of simulating realistically configured large-scale

commercial applications, we also have the additional problem of simulating an enormous CMT design space. Thus, we were confronted by the canonical simulation trade-off between absolute accuracy and simulation time. While we could have simulated scaled-down versions of the commercial applications running for short periods of time on a detailed, execution-driven simulator, the variability in IPC of such an approach would have been too high [2]. Instead, we employ the RASE (Rapid, Accurate Simulation Environment) approach, a collection of Sun Microsystems' internal tools, that uses a highly detailed and cycle-accurate execution-driven *and* trace-driven simulation to characterize the CMT designs by generating multi-configuration instruction trace files. RASE uses SimCMT, a configurable CMT timing simulator built on top of Simics by VirtuTech [18] for execution-driven simulation similar to the methodology used in Simflex and GEMS [8][20]. SimCMT can also be used as a fast stand-alone trace-driven simulator.

### 3.1. Simulation methodology

RASE does not scale down the application, but instead runs large applications in steady state on both real system hardware and on SimCMT. We have correlated and validated the execution-driven model against real hardware, both SMPs and CMTs. During this correlation, system hardware counters are compared to simulated counters for primary and secondary cache, and TLB latencies and miss rates. In addition, comparisons of instruction mixes, context switch frequencies, intervals between contexts, and so on, are correlated to ensure accurate reproduction at the workload level.

We generate multi-configuration instruction traces using execution-driven simulation that can be pre-processed and then used in a fast, accurate trace-based simulation mode of SimCMT. While there is substantial cost and effort up-front in the RASE methodology, once that cost has been paid, RASE has several advantages over execution-driven simulation of scaled commercial applications. The trace-driven simulation enables faster simulation of various configurations compared to an execution-driven model and the lack of variability in the test sequence isolates the effects of architectural changes. In addition, using large instruction trace files in trace-driven simulation for long time periods addresses the issue of nonrepeatability in commercial applications across multiple short runs [1][2], at the cost of simulating instruction sequences that might not be possible in an actual multithreaded execution. We have observed approximately 1% difference in IPC when comparing

execution-driven and trace-driven full system simulations for the same CMT model for TPC-C and SPEC JBB. As expected, miss rates for the two methodologies differ more, but we have not observed a relative difference greater than 5%. The downside to the RASE methodology is that for the initial correlation, one must have access to a full-size commercial machine, which can cost millions of dollars.

### 3.2. Benchmark details

We selected SPEC JBB, TPC-C, TPC-W, and XML Test server benchmarks to assess the CMT's performance. SPEC JBB emulates a 3-tier system emphasizing the Java server-side performance of middleware business logic [30]. TPC-C is an online transaction processing benchmark based on an order-entry system [31]. We concentrate on the server component of TPC-C for this study. This complicated benchmark has extreme hard disk, memory, and network resource requirements [14][4][1][27]. TPC-W is a transactional web benchmark that simulates the activities of a business oriented transactional web server [31]. XML Test is a multithreaded XML processing test developed at Sun Microsystems [32]. XML Test performs both streaming and tree-building parsing, which replicate application servers that provide web services and simultaneously process XML documents. Unlike SPEC JBB, XML Test is a single tier system benchmark; the test driver is integrated into worker thread.

These benchmarks do not exhibit multiphase execution, so recording contiguous streams of instruction on a per thread basis can capture the complete system performance, the overall benchmark characteristics, *and* the instruction mix. In contrast, benchmarks like SPEC CPU2000 require sampling techniques to capture the various phases of execution [24]. SPEC JBB uses the J2SE 1.4 JVM with a 2 GB heap running on Solaris 9 with 16 warehouses to collect a 16-processor instruction trace file. XML Test uses the J2SE 1.5 JVM, but with a 2.5 GB heap for a 16-processor trace file. For TPC-C, we use 3,000 warehouses with a 28 GB SGA and 176 9 GB disks coupled with commercial database management and volume manager software running on Solaris 9. For both TPC-C and TPC-W, the clients and servers are simulated, but only the server instruction traces are used in this study. TPC-W can support up to 10,000 users. The database is built on 28 9 GB disks coupled with commercial database management and volume manager software running on Solaris 9. The application server uses JDK 1.4.x, while JDK 1.3.x is used for the image server, payment gateway emulator, and the SSL

**Table 2: CMT design space parameters segmented (alternating gray areas) to indicate major core configuration groups. All 4 L2 cache configurations are used with all core configurations per class.**

CMT Scale				Small			Medium			Large		
Core Config	Number of IDPs	Number of Threads	Max L1 Size (KB)	L2 Cache (MB, SA)	Number of Processors	Aggregate Threads	L2 Cache (MB,SA)	Number of Processors	Aggregate Threads	L2 Cache (MB,SA)	Number of Processors	Aggregate Threads
1p2t	1	2	32	1, 16	4-11	8-22	1.5, 12	5-20	10-40	3,24	10-34	20-68
1p4t	1	4	32		3-10	12-40		5-17	20-68		8-30	32-120
1p8t	1	8	64		2-8	16-64		3-14	24-112		7-25	56-200
2p2t	2	2	32/64		3-9	6-18		4-16	8-32		8-28	16-56
2p4t	2	4	64		2-8	8-32		3-14	12-112		7-25	28-100
2p8t	2	8	64/128		2-6	16-48		3-12	24-96		5-21	40-168
2p16t	2	16	128		1-5	16-80		2-9	32-144		4-15	64-240
3p3t	3	3	64		2-7	6-42		3-13	9-39		6-22	18-66
3p6t	3	6	64/128		1-6	6-36		3-11	18-66		5-20	30-120
3p12t	3	12	128		1-5	12-60		2-9	24-108		4-15	48-180
3p24t	3	24	128		1-3	24-72		1-6	24-144		3-10	72-240
4p8t	4	8	64/128		1-5	8-40		2-9	16-72		4-15	32-120
4p16t	4	16	128	1-3	16-48	2-7	32-112	3-10	48-160			
2s1t	2	1	64	2-6	2-6	4-11	4-11	7-18	7-18			
2s2t	2	2	64	2-5	4-10	4-10	8-20	6-17	12-24			
2s4t	2	4	64	2-5	8-20	3-9	12-36	6-15	24-60			
2s8t	2	8	64	1-4	8-32	2-7	16-56	5-12	40-96			
4s1t	4	1	64	1-4	1-4	2-7	2-7	4-11	4-11			
4s2t	4	2	64	1-3	2-6	2-6	4-12	4-10	8-20			
4s4t	4	4	64	1-3	4-12	2-5	8-20	3-9	12-108			
4s8t	4	8	64	1-3	8-24	1-4	8-32	3-7	24-56			

components. Fixed processor sets are used to isolate the application servers from the rest of the simulation, allowing us to harvest the instruction streams only from the application server processor set.

Each trace contains several billion instructions per process thread in steady state. All traces are collected during the valid measurement time after the benchmarks have ramped up and completed the benchmark specified warm-up cycle, as on real hardware. We have observed significant variation in benchmark performance during the ramp-up period, but little variation once in steady state, as observed in [2]. All benchmarks are highly tuned, with less than 1% system idle time, and show negligible performance variability during the measurement period.

### 3.3. CMT architecture design space

Table 2 summarizes the parameter ranges that we investigated, subject to the constraints on the processor cores and processor/secondary cache die division from Section 2. The maximum primary cache capacities are shown as a single value or as  $X/Y$  if the maximum is asymmetric, where one L1 cache is larger than the other. For this latter case, the set associativity of the larger cache in the asymmetric pair remained low to further constrain the area. Finally, for each CMT class (small, medium, or large), there are four secondary cache sizes for each of the 21 core configurations, corresponding to approximately 25%, 40%, 60%, and 75% of the CMT area. To prevent the DRAM

bandwidth from becoming a bottleneck, we chose an aggressive but achievable number of DRAM controllers/channels for our design. We use 4 dual Fully Buffered (FB)-DIMM DRAM channels shared by the 8 banks of the shared secondary cache, 8 dual FB-DIMM DRAM channels shared by 8 cache banks, and 16 dual FB-DIMM DRAM channels shared by 16 banks for small, medium, and large scale CMTs, respectively.

The in-order scalar and superscalar cores utilize fully pipelined integer and floating-point datapaths, with each datapath capable of executing one instruction per cycle. Each processor core consists of one to four integer datapath pipelines (IDPs or integer ALUs). Up to 8 hardware threads are supported per IDP within the processor core, while up to 8 hardware threads are supported per superscalar processor core. The nomenclature we use to label the scalar cores is  $NpMt$ , where  $N$  is the number of IDPs in the core, and  $M$  is the total number of hardware threads supported by the core. We differentiate the scalar cores from the superscalar cores by labeling them  $NsMt$ , where  $N$  denotes the issue width of the superscalar processor. Each scalar integer pipeline can only execute instructions from a statically assigned pool of  $M/N$  threads, whereas the superscalar pipelines can issue instructions from any of  $M$  threads. Each core contains a single-ported primary data and instruction cache shared between the IDPs, sized from 8 KB up to the values shown in Table 2.

Fine-grain multithreading [16][29] is used in our cores to address the low ILP and high cache miss rates of commercial workloads. Simultaneous multithreading

(SMT) is the fine-grain multithreading technique of choice being used in modern superscalar processors such as the hyperthreaded Intel processors [19] or the Power5™ [10], and has been shown to give good performance benefits for commercial workloads [17]. SMT interleaves execution from multiple hardware threads across both vertical (processor cycle) and horizontal (issue width) dimensions [29]. For scalar processors, only the vertical thread interleaving is applicable, and thus multithreading a scalar processor by switching threads every cycle has been labeled as vertical or interleaved multithreading [16]. Our superscalar processor instruction scheduling policy is based on selecting as many instructions as possible from the threads in LRU order. The multithreading employed by our scalar processors is very similar to that described in [16], where instructions that have long-latency but determinant execution times (e.g. floating-point operations) are detected early in the pipeline (at decode) and prevent a thread from issuing further instructions until the long-latency operation result is available, while instructions that have an indeterminate execution time (e.g. loads) do not prevent issuing of further instructions to the pipeline. Instead, these indeterminate-latency instructions cause a selective pipeline flush of all instructions from the thread when it is discovered that they are indeed a long-latency operation.

The processor core clock frequency, as shown in Table 1, is chosen to be sufficient to allow an individual thread to comfortably meet the latency requirements of our commercial applications, while at the same time producing a CMT with just half of the power dissipation of a standard high-frequency monolithic server processor, making it much more suitable for use in a dense server environment.

## 4. Results

The CMT design space is too large to be fully characterized by simulation. Industry experience, intuition, and literature surveys [15] provided initial guidance for creating a tractable CMT design space. We then performed a two-phase simulation study. The first phase, for which we do not present data, used a perfect secondary cache model; all primary cache misses were secondary cache hits. These simulations provided a theoretical upper bound for the processor core IPC. It also provided insight into L1 cache performance based on size and set associativity. The second phase used a detailed secondary cache and Fully Buffered DIMM (FB-DIMM) memory subsystem model.

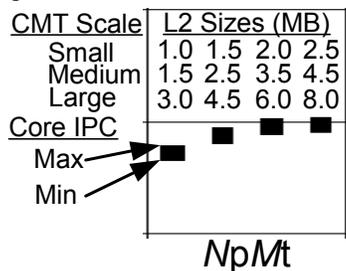
The commercial server applications exhibited a range of low to moderate ILP and high cache miss rates similar to the observations in [14]. Using a single thread per pipeline provides no hardware mechanism for latency tolerance and results in low processor utilization, or “underthreading.” On the other hand, too many active threads can lead to an “overthreaded” core with a fully utilized integer datapath pipeline (IDP) and performance that is insensitive to primary cache capacity or set associativity. Our goal was to find a good balance that optimized aggregate IPC and remained general purpose. Thus, we removed single-thread cores from future study (except for 2p2t and 3p3t, which we included in our study as underthreaded examples), as they could not conceal the long latencies common in these applications. We also removed all cores with more than 8 threads per IDP because 8 or more threads saturated the integer pipeline. We also pruned the 4p32t, 4 pipelines with 32 threads per core, and all superscalar core configurations with more than 8 threads due to area and complexity constraints.

### 4.1. Processor core performance

All cores within the CMT are simulated and the average IPC across all the cores is shown in Figure 4 for medium-scale CMTs executing SPEC JBB. Figure 3 illustrates how to interpret the data in Figure 4. Each vertical column of the graph represents one of the four secondary cache configurations, increasing in size from left to right, for a given processor core configuration (NpMt or NsMt) labeled below that segment along the x-axis. Each black bar defines the IPC range, min (bottom) and max (top), for all primary cache size pairs and set associativities for that particular secondary cache size and processor core configuration. In Figure 3, the secondary cache capacities are labeled above each bar for small to large-scale CMTs. Figure 4 shows the range (maximum to minimum) of the average processor core IPC (y-axis) derived from the full CMT results for a particular configuration.

These simulations correlated with our intuitive performance trends based on primary cache size and issue width of the processor. The superscalar CMT cores outperform their scalar counterparts, i.e., 2p4t vs. 2s4t, but as we will see in Section 4.2, the increased core area prevents the superscalar CMTs from outperforming their scalar counterparts at a full-chip system level. With regard to primary cache size, more is better, not surprising given that we assume no cycle time penalty from larger primary and secondary caches. For example, the maximum core IPC for the medium-scale 1p4t SPEC JBB is a configuration with the largest possible primary data and instruction cache and

secondary cache defined in Table 2. This same core configuration coupled with the smallest secondary cache results in 15% performance degradation in core IPC. Note that the 1p8t configurations with the larger secondary cache configurations in Figure 4 are examples of “overthreading with a pipeline utilization over 94% and a very small average core IPC range. This configuration exhibits similar performance behavior for all the benchmarks; it is insensitive to primary cache size and set associativity as a result of “overthreading”.

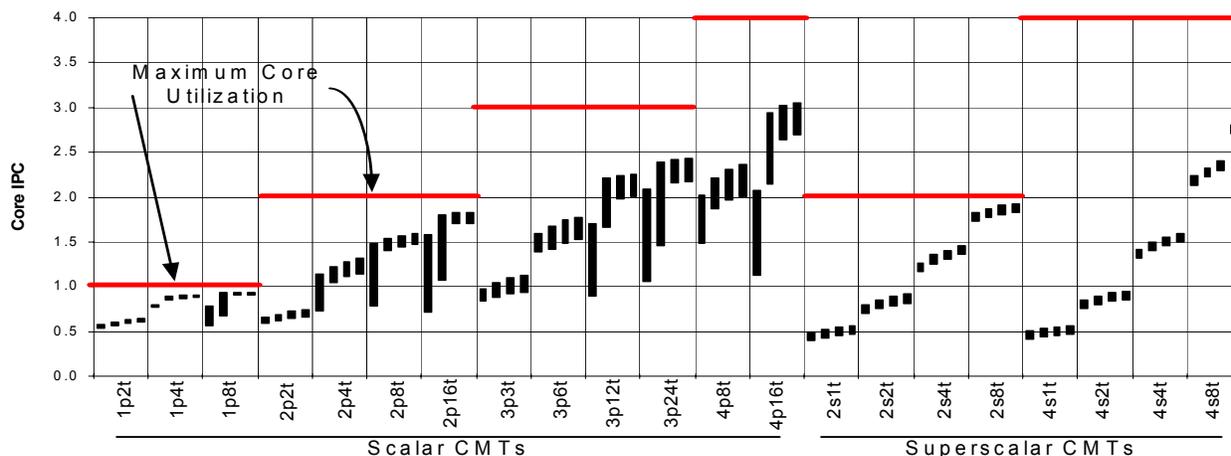


**Figure 3: Average core IPC range for the  $NpMt$  configuration for 4 different L2 cache sizes. Each black bar shows the IPC range when L1 cache sizes are varied for each L2 cache size.**

This pipeline saturation or “overthreading” can easily be observed in the 1p8t, 2p16t, and 2s8t core configurations, where the maximum IPC is very close to the absolute peak. Figure 4 also illustrates the performance degradation as a result of insufficient secondary cache capacity. This is most noticeable in the large IPC degradation or step down for the 2p16t, 3p12t, 3p24t, or 4p16t configuration when moving right to left from a 2.5 MB L2 to a 1.5MB L2, where the small primary cache configurations magnify the effects of insufficient secondary cache capacity. This problem is exacerbated in the medium-scale CMTs due to the

increased number of aggregate threads for core-intensive designs, but is also present in the small-scale CMT configurations with large numbers of IDPs and threads. Given the memory subsystem scaling, we have observed that the (limited) large-scale CMT results exhibit performance characteristics similar to the small-scale and medium-scale CMT results. In addition to insufficient secondary cache capacity, SPEC JBB can suffer from insufficient secondary cache associativity as well. In Figure 4, insufficient secondary cache associativity degrades the performance of both large and small cores for the core-intensive configurations with 8 or more threads per core. In these cases, conflict misses in the secondary cache cause serial thread execution by forcing threads to wait on main memory accesses. This can be further aggravated if the same secondary cache bank and/or DRAM bank become memory hot spots [26].

One of the benefits of multithreading is its ability to tolerate latency, but there are conditions caused by thread interference that saturate the memory bandwidth, negating this ability to hide latency and causing the performance of the processor core to drop dramatically. This is best illustrated by the large CMT configurations (3p13t, 2p16t, 3p24t, and 4p16t) with small L2 and large average IPC ranges. The medium-scale CMT results in Figure 4 are similar for small-scale and large-scale CMTs on all the benchmarks. In general, the reader can scale the average IPC in Figure 4 up or down depending on the benchmark, up for XML Test and down for TPC-W and TPC-C. The “overthreaded” configurations saturate the pipelines for the same CMT configurations across all the benchmarks. TPC-C’s performance is more sensitive to secondary cache size and results in a more pronounced performance “step” traversing the increasing secondary cache sizes. TPC-



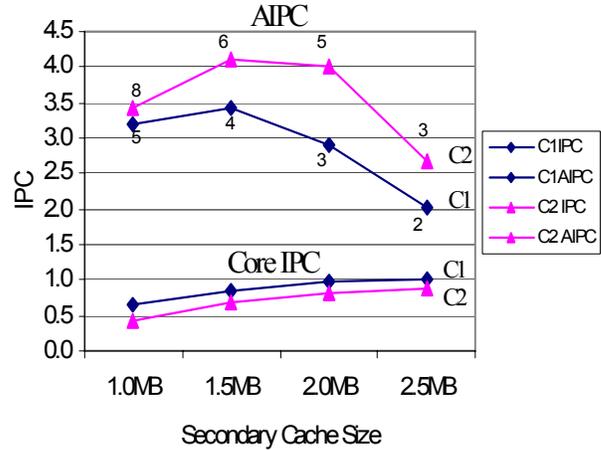
**Figure 4: SPEC JBB average core IPC range (maximum to minimum) for medium-scale CMTs. The secondary cache size range is 1.5MB, 2.5MB, 3.5MB, and 4.5MB from left to right for each core.**

W has slightly higher average core IPC and exhibits similar performance to TPC-C. XML Test has the highest average core IPC.

### 4.2. CMT performance

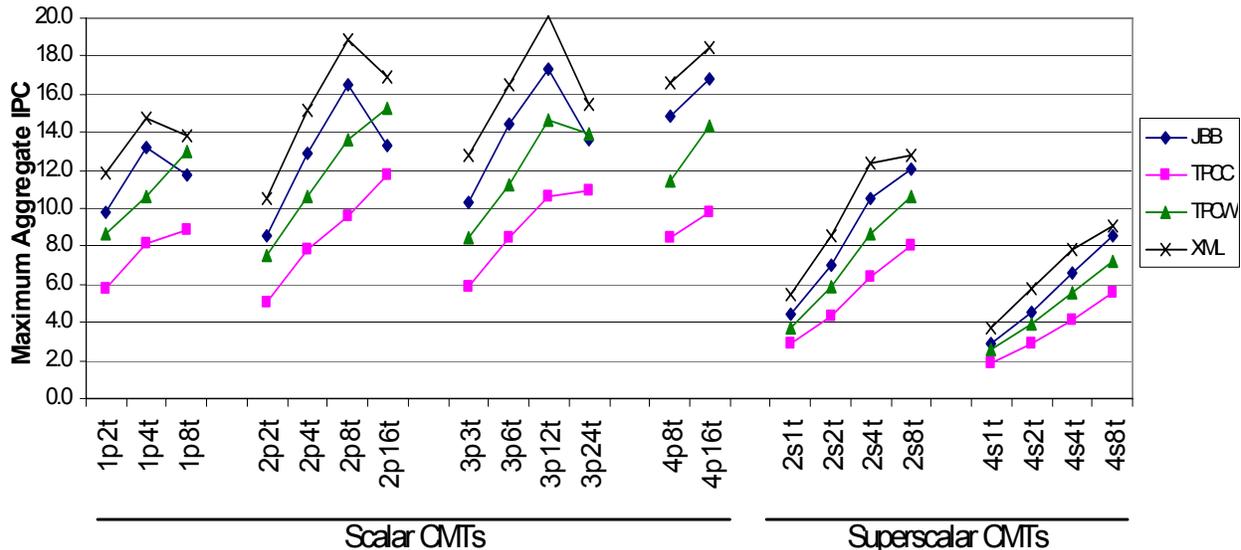
Historically, the goal of optimizing the processor core was to squeeze out every last percent of performance that can be achieved with reasonable area costs. However, in the CMT design space, this is a local optimization that does not yield high aggregate performance. This is exemplified by the aggregate IPC results for the 2p4t core configuration shown in Figure 5. The top two lines are the aggregate IPC's (AIPCs) for a particular cache configuration and the bottom two lines are the corresponding average core IPC's. C1 represents the 2p4t configuration with the best core IPC, 64KB data and instruction cache, but its corresponding AIPC underperforms due to the small number of cores that can be fit on the die. On the other hand, C2 is a "mediocre" 2p4t configuration with only a 32 KB data and instruction cache, but it has the best AIPC by maximizing the number of cores for a given secondary cache size, as indicated in Figure 5. C2 also illustrates that too many cores on the chip can degrade overall performance. As both the total number of cores that can be fit on the chip and the performance of each of those cores are strongly dependent on the amount of on-chip secondary cache, it is important to balance processing and cache needs. We present the best results for each core configuration and all of the benchmarks used in this study in Figure 6 for the medium-scale CMTs. This figure provides the maximum AIPC (y-axis) across all cache configurations for all pipeline/thread configurations (x-axis). The number of cores and

cache configurations that yield the AIPC in Figure 6 is provided in Table 3 for each pipeline/thread configuration. The CMTs are clustered by pipeline and pipeline architecture, scalar vs. superscalar.



**Figure 5: Small-scale CMT TPC-C core and aggregate IPC for the 2p4t CMT configuration. C1 has the best average core IPC. C2 has the best aggregate IPC by using more cores on the die. The number of cores for each CMT is labeled next the upper pair of lines.**

Table 3 shows the maximum AIPC for SPEC JBB, TPC-C, TPC-W, and XML Test for medium-scale CMTs. This table lists the best configuration for each core configuration and highlights the overall best CMT configuration in black boxes. The AIPC scales proportionally with the number of cores. Thus, the reader can derive the omitted (due to space constraints) small-scale and large-scale CMT results from the medium-scale results in Table 3.



**Figure 6: Medium-scale CMT aggregate IPC for each CMT configuration and all benchmarks.**

**Table 3: Maximum AIPC for medium-scale CMTs for SPEC JBB, TPC-C, TPC-W, and XML Test.**

Core Config	SPEC JBB 2000				TPC-C				TPC-W				XML Test			
	L1	L2	Cores	AIPC	L1	L2	Cores	AIPC	L1	L2	Cores	AIPC	L1	L2	Cores	AIPC
1p2t	16/32	1.5/12	20	9.8	16/32	2.5/10	16	5.8	16/32	1.5/12	20	8.6	16/32	1.5/12	20	11.8
1p4t	16/32	1.5/12	17	13.2	16/32	2.5/10	14	8.2	16/32	1.5/12	17	10.6	16/32	1.5/12	17	14.8
1p8t	16/32	2.5/10	12	11.7	32/32	1.5/12	14	8.9	32/32	1.5/12	14	13.0	16/32	1.5/12	14	13.8
2p2t	16/32	1.5/12	16	8.6	16/32	1.5/12	16	5.1	16/32	1.5/12	16	7.5	16/32	1.5/12	16	10.5
2p4t	32/32	1.5/12	14	12.9	32/32	2.5/10	12	7.8	32/32	1.5/12	14	10.6	16/32	1.5/12	14	15.2
2p8t	16/32	1.5/12	12	16.5	32/32	2.5/10	9	9.5	32/32	1.5/12	12	13.6	32/32	1.5/12	12	18.9
2p16t	32/64	2.5/10	7	13.3	64/64	2.5/10	7	<b>11.8</b>	64/64	1.5/12	9	<b>15.2</b>	32/64	1.5/12	9	16.9
3p3t	32/32	1.5/12	13	10.3	32/32	2.5/10	10	5.9	32/32	1.5/12	13	8.5	16/32	1.5/12	13	12.7
3p6t	32/32	1.5/12	11	14.4	32/32	2.5/10	9	8.5	32/32	1.5/12	11	11.3	32/32	1.5/12	11	16.5
3p12t	32/64	1.5/12	9	<b>17.3</b>	32/64	2.5/10	7	10.7	64/64	1.5/12	9	14.6	32/64	1.5/12	9	<b>20.1</b>
3p24t	32/64	2.5/10	5	13.6	32/64	2.5/10	5	10.9	32/64	1.5/12	6	14.0	32/64	1.5/12	6	15.5
4p8t	32/32	1.5/12	9	14.9	32/32	2.5/10	7	8.5	64/64	1.5/12	9	11.5	16/32	1.5/12	9	16.6
4p16t	32/64	1.5/12	7	16.8	32/64	2.5/10	5	9.8	64/64	1.5/12	7	14.4	32/64	1.5/12	7	18.5
2s1t	64/64	1.5/12	11	4.4	64/64	1.5/12	11	2.8	64/64	1.5/12	11	3.7	64/64	1.5/12	11	5.5
2s2t	64/64	1.5/12	10	7.0	64/64	1.5/12	10	4.3	64/64	1.5/12	10	5.8	64/64	1.5/12	10	8.6
2s4t	64/64	1.5/12	9	10.5	64/64	1.5/12	9	6.4	64/64	1.5/12	9	8.7	64/64	1.5/12	9	12.4
2s8t	64/64	1.5/12	7	12.1	64/64	1.5/12	7	8.1	64/64	1.5/12	7	10.6	64/64	1.5/12	7	12.7
4s1t	64/64	1.5/12	7	2.9	64/64	1.5/12	7	1.9	64/64	1.5/12	7	2.6	64/64	1.5/12	7	3.7
4s2t	64/64	1.5/12	6	4.5	64/64	1.5/12	6	2.9	64/64	1.5/12	6	3.9	64/64	1.5/12	6	5.8
4s4t	64/64	1.5/12	5	6.6	64/64	1.5/12	5	4.1	64/64	1.5/12	5	5.6	64/64	1.5/12	5	7.8
4s8t	64/64	1.5/12	4	8.5	64/64	1.5/12	4	5.5	64/64	1.5/12	4	7.2	64/64	1.5/12	4	9.1

Note: The L1 refers to the primary data/instruction cache size. The L2 cache configuration size (MB)/set associativity (SA) are provided along with the total number of cores for that CMT configuration.

### 4.3. Discussion

We have shown that augmenting CMPs with fine-grain multithreading, creating CMTs, is crucial to increasing the performance of commercial server applications. Furthermore, scalar CMT variants with 4 or more threads outperformed nearly all of the superscalar CMT configurations given the constant die size constraint. While multiple processor cores can exploit TLP, fine-grain multithreading is also necessary to alleviate the otherwise poor core utilization for these applications. However, we found that fine-grain multithreading runs into two limits. First, the addition of too many threads results in a saturated integer pipeline that was insensitive to L1 cache parameters. In our studies, we found that this saturation occurred with about 8 threads per integer pipeline for scalar cores. Second, a CMT built with too many total threads for the secondary cache size can end up saturating the memory bandwidth with secondary cache misses, as the aggregate working set overflows the secondary cache. We encountered memory saturation primarily for the configurations that had the smallest secondary cache size (occupying 24-28% of the CMT area) and 8 or more threads per core. We found aggregate IPC to be optimized by a processor centric design, requiring only 25-40% of the area devoted to the shared secondary cache. When focusing

on the processor core itself, a larger primary instruction cache than the primary data cache is always the best policy. Surprisingly, high primary cache set associativity was not required for these applications, even with more threads than set associative ways.

For a given primary data and instruction cache configuration, the performance difference based on set associativity varied less than 3% for the best aggregate IPC configurations, as long as the caches were at least two-way set associative. We also found that the best performing configurations required enough threads and primary cache to bring the pipeline utilization up to the 60-85% range, as the area costs for adding additional pipelines and threads per pipeline is much smaller than adding an additional core. For small, medium, and large-scale CMTs, the best configuration was with 3 pipelines and 12 threads per core for Spec JBB amd XML Test, while 2 pipelines and 16 threads per core performed best for TPC-C and TPC-W. We also found that the best performing CMT configuration was highly dependent on a step function of the number of cores that can be squeezed on the die, allowing a CMT composed of slightly lower performance cores to yield superior aggregate performance by employing more of those cores. As a corollary to this step function regarding core size, processor cores with smaller primary caches were favored, even without penalizing

the larger caches with additional latency, as the smaller-cache cores maximized the number of on-chip cores.

Interestingly enough, for small, medium, and large-scale CMTs, the worst performing configurations for SPEC JBB and XML Test included both “underthreaded” and “overthreaded” configurations, while the worst performing configurations for TPC-C and TPC-W were always “underthreaded” configurations. This matches our intuition, as the low IPC of a single TPC-C or TPC-W thread makes underthreading more detrimental, while the more moderate IPC of a single SPEC JBB or XML Test thread makes it more susceptible to both underthreading and overthreading. Finally, 2-way superscalar configurations outperformed all 4-way superscalar configurations with the same number of threads.

#### 4.4. Technology trends

One of our assumptions is that a large, near reticle limit die will be used for the CMT chip. However, even with a mature silicon process, wafer yield for a reticle-limited die is extremely low. Thus, we note that our results apply to a 225-250 mm<sup>2</sup> chip in the subsequent process generation, which is much more manufacturable and affordable. The results we presented for small-scale, medium-scale, and large-scale CMTs could then map to the 90 nm, 65 nm, 45 nm process generations, respectively. Furthermore, adding more SOC components or engines beyond the base CMT would promote a similar process generation “migration,” even with a larger die. Regardless of the reasons for using the next silicon process generation, the results are still valid because the clock frequencies for both on-chip and off-chip components scale proportionally if the clocking rules in Table 1 are used. For the highest end servers, however, an expensive, reticle-limited die may still be appropriate, as the cost of these servers are often dominated by components other than the processor (e.g. memory).

FB-DIMM memory provides sufficient bandwidth for CMT designs in 130 nm, 90nm, and 65 nm generations, but pin pressure for the FB-DIMM interface reappears when we continue to double the memory bandwidth for the 45nm process technology. As DDR2 is the memory technology of choice for the 130nm generation, we also simulated a more bandwidth constrained 4-channel DDR2 memory system for the small-scale CMT designs. The DDR2 results differed from the FB-DIMM results in that more on-chip cache was optimal (40-60% of the area instead of 25-40%), and the penalty for being “overthreaded” was more pronounced. When examining the amount of secondary cache memory per thread, we observed that for SPEC

JBB 15 KB/thread, TPC-C 25 KB/ thread, and TPC-W and XML TEST less than 10 KB/thread was required to prevent severe performance degradation using the FB-DIMM memory interface. This secondary cache memory requirement is doubled when using a DDR memory interface. Given the likelihood of future insufficient memory bandwidth, we believe that CMT designs in 45 nm and beyond will be skewed towards a more equal balance in the area between the upper level caches and the processor cores. However, there are techniques like cache compression described in [3] that counteract the effects of memory bandwidth scaling and the resulting pin pressure and thereby would shift the optimal CMT configurations back toward core-centric as a result of increasing on-chip cache efficiency.

#### 5. Related work

Scaling superscalar processors leverage existing intellectual property (IP) and reduce initial design effort. However, replicated hardware or other evolutionary, ILP-focused industry solutions [19][10][11] are not necessarily ideal for commercial server applications. Superscalar processors require large issue widths and/or large number of threads to achieve a small fraction of their peak performance on commercial workloads [14][4][27]. This is likely to lead to low aggregate performance in a CMP due to the small number of superscalar cores that will fit on a die. [12] also leverages existing IP and presents a performance evaluation of a heterogeneous CMP. This architecture study focuses on a CPU-intensive diverse workload instead of the commercial server homogeneous workloads that we use.

A large body of research exists that is concerned with optimizing the commercial application space and the microprocessors that execute these programs [3][5][7][12][17][22][27]. SMT is one possible solution to hiding long latency events and high cache miss rates. Most of industry appears to be heading down this path [6][19], but we believe that the area overhead and low utilization of the superscalar processor cores will ultimately result in a CMT of lower aggregate performance than one composed of simple cores, similar to the approach described in [13]. In addition, both Intel [19] and IBM [10] have stated the limits of SMT to be two threads, and it is clear from our results that two threads are not sufficient to mask the long latency events in all of these benchmarks.

[4] and [27] provided motivation for using simple core CMPs for commercial server applications. The Piranha project investigated using CMPs with simple, single-issue in-order cores running OLTP and DSS workloads, TPC-B and TPC-D [5], compared to

monolithic superscalar processors. The Piranha processors outperformed the monolithic cores, but lacked the key feature of fine-grained multithreading to mask memory latency inside the processor core, which this study has demonstrated to be critical to achieve high aggregate CMT performance. Piranha was also a research prototype targeting the 180 nm generation, and as such selected a particular CMT design point (eight 1p1t 64 KB instruction and data caches with a 1MB secondary cache) [5] to compare against a monolithic superscalar processor of similar die size. Our study greatly expands on the Piranha work by not only exploring the addition of fine-grain multithreading to simple cores and comparing scalar and superscalar CMTs, but also by exploring multiple technology generations, cache configurations, and a significant portion of this CMT design space, all while running large scale full system simulations of SPEC JBB, TPC-C, TPC-W and XML Test.

[9] presents an exploration of CMPs built from either in-order or out-of-order superscalar processors, but differs from our work in many ways. [9] uses a different class of benchmarks (SPEC CPU2000), uses partial CMP simulation to extrapolate CMP performance, does not examine either multithreaded or scalar cores, and provides a private L2 cache per core, which greatly increases data sharing overhead. They also use performance-scaling techniques that overlook the memory saturation issues that we encountered. This is even more relevant because their area model predicts higher core counts for CMPs in the same process generation, which greatly increases memory bandwidth requirements. Furthermore, there is no mention of memory coherence, which is required for this application domain. In contrast to [9], we use large scale applications from the target domain, perform full system simulations of all CMT configurations, both scalar and superscalar, examine a CMT architecture with a large shared secondary cache to exploit data sharing, and maintain full memory coherence. Our study points toward CMTs built from small scalar cores as performing best for commercial workloads, while their study pointed towards CMPs built from large out-of-order superscalar cores as performing best, unless the application was bandwidth bound.

## 6. Conclusions

In this paper, we explored the performance of multithreaded scalar and superscalar core CMTs on commercial workloads for small, medium and (to a limited extent) large-scale systems. When comparing area-equivalent scalar and superscalar CMT configurations, we found that scalar CMTs with small

primary caches significantly outperform their superscalar counterparts by 37-46%. Even though the superscalar processors achieve a higher core IPC than the scalar processors, the increased number of small scalar cores that can be fit on a die more than makes up for this difference. This ability of “mediocre” cores to provide the best aggregate performance on commercial workloads is a key contribution of this throughput study. Our study showed multithreading was also crucial to achieve good application performance; however, too many threads led to execution pipeline saturation or, in the extreme case, to memory bandwidth saturation. This is counter to the multithreading efficiency limits of 2 threads stated by [10] and [19]. However, our design is targeting a CMT with a high-bandwidth memory subsystem, which is crucial to being able to keep the large number of threads fed.

We also found that the best performing configuration was highly dependent on a step function of the number of cores that could be squeezed on the die. As a result, processor cores with smaller primary caches were favored, even without penalizing the larger caches with additional latency, as the smaller-cache cores maximized the number of on-chip cores. Table 4 shows the optimal small, medium, and large-scale CMT configurations. While one single configuration did not perform optimally for all of the benchmarks, our results show a range, usually using 4-8 threads per pipeline, of high performing CMT configurations.

**Table 4: Maximum AIPC for all benchmarks.**

Benchmark	Core	Small Scale Cores, AIPC	Medium Scale Cores, AIPC	Large Scale Cores, AIPC
SPEC JBB2000	3p12t	5, 9.6	9, 17.3	15, 30.8
TPC-C	2p16t	5, 6.4	7, 11.8	12, 20.8
TPC-W	2p16t	5, 8.3	9, 15.2	15, 27.7
XML Test	3p12t	5, 11	9, 20.1	15, 35.4

Our results show that it is necessary to perform full system simulation to achieve accurate performance estimates. Basing performance predictions on scaled or partial systems simulations would not have revealed the dramatic performance drop-off for “overthreaded” configurations, which saturate the memory subsystem. We purposely assumed each CMT had an aggressive memory subsystem, but this interface still became saturated for large numbers of threads and/or cores. We were surprised that multiple scalar pipelines sharing a single instruction and single data cache port was not a source of significant performance degradation, even with instruction buffers for each thread. Likewise, both the primary and secondary caches were relatively insensitive to variations in set associativity, regardless of capacity or the number of supported threads. In this study, we have temporarily circumvented the memory wall by reducing processor frequency for CMT

configurations and by scaling DRAM interface frequencies over time. Eventually future physical constraints, such as limited package pins in the 45 nm generation, will present a challenge to future computer architects by limiting off-chip bandwidth for commercial server benchmarks.

## Acknowledgements

We would like to thank Cong Fu, Venkatesh Iyengar, and the entire Niagara Architecture Group for their assistance with the performance modeling. We would like to thank Lance Hammond, Todd Austin, and the anonymous reviewers for their comments. This research was supported by Sun Microsystems and National Science Foundation grant CCR-0220138.

## References

- [1] A.R. Alameldeen, C. J. Mauer, et al., "Evaluating Non-deterministic Multi-threaded Commercial Workloads," Computer Architecture Evaluation using Commercial Workloads (CAECW), February 2002.
- [2] A.R. Alameldeen and D.A. Wood, "Variability in Architectural Simulations of Multi-threaded Workloads," 9th Int'l Symp. on High Performance Computer Architecture (HPCA), February 2003.
- [3] A.R. Alameldeen and D.A. Wood, "Adaptive Cache Compression for High-Performance Processors," Proc. of the Annual Int'l Symp. on Computer Architecture (ISCA) 2004.
- [4] L. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads," ISCA-25, June 1998, pages 3-14.
- [5] L. Barroso, K. Gharachorloo, R. McNamara, et al., "Piranha: a scalable architecture based on single-chip multiprocessing," ISCA-27, June 2000, pages 282-293.
- [6] J. Clabes, J. Friedrich, and M. Sweet, "Design and Implementation of the POWER5™ Microprocessor," ISSCC Dig. Tech. Papers, pages 56-57, Feb. 2004.
- [7] F. Eskesen, M. Hack, T. Kimbrel, et al., "Performance Analysis of Simultaneous Multithreading in a PowerPC-based Processor," IBM Research Report, May 2002, RC22454.
- [8] N. Hardavellas, S. Somogyi, et al., "Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture," *SIGMETRICS Performance Evaluation Review*, pages 31-35, April 2004.
- [9] J. Huh, S.W. Keckler and D. Burger, "Exploring the Design Space of Future CMPs," Int'l Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 199-210, Sept. 2001.
- [10] R. Kall, B. Sinharoy, J. Tandler, "SMT Implementation in Power5," 15th Hot Chips Symp., August 2003.
- [11] S. Kapil, "Gemini: A Power-efficient Chip Multi-Threaded UltraSPARC® Processor," 15th Hot Chips Symp., August 2003.
- [12] R. Kumar, D. Tullsen, et al., "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance," ISCA-31, June 2004.
- [13] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way Multithreaded Sparc Processor," IEEE Micro, pages 21-29, March/April 2005.
- [14] S. Kunkel, R. Eickemeyer, M. Lipasti, T. Mullins, "A performance methodology for commercial servers," IBM Journal of Res. and Dev., Vol. 44, Number 6, 2000.
- [15] F. Labonte and M. Horowitz, "Microprocessors Through the Ages," [http://www-vlsi.stanford.edu/group/chips\\_micropro.html](http://www-vlsi.stanford.edu/group/chips_micropro.html)
- [16] J. Laudon, A. Gupta, and M. Horowitz, "Interleaving: A Multithreading Technique Targeting Multiprocessors and Workstations," Proc. of the 6th Int'l Symp. on Architectural Support for Parallel Languages and Operating Systems (ASPLOS), October 1994, pages 308-318.
- [17] J. Lo, L. Barroso, S. Eggers, K. Gharachorloo, et al., "An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors," ISCA-25, Jun 1998, pages 39-50.
- [18] P. Magnusson, M. Christensson, J. Eskilson, et al., "Simics: A Full System Simulation Platform," Computer, February 2002, pages 50-58.
- [19] D. Marr, "Hyper-Threading Technology in the Netburst® Microarchitecture," 14th Hot Chips, August 2002.
- [20] M. Martin, D. Sorin, B. Beckmann, et al., "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset," Computer Architecture News (CAN), 2005.
- [21] K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The Case for a Single-Chip Multiprocessor," ASPLOS-7, October 1996.
- [22] P. Ranganathan, K. Gharachorloo, S. Adve, and L. Barroso, "Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors," ASPLOS-8, Oct. 1998, pages 307-318.
- [23] Semiconductor Industry Association (SIA), International Technology Roadmap for Semiconductors (ITRS), 2001 edition with 2002 update.
- [24] T. Sherwood, S. Sair, and B. Calder, "Phase Tracking and Prediction," ISCA-30, June 2003.
- [25] P. Shivakumar, N. Jouppi, "CACTI 3.0: An Integrated Cache Timing, Power, and Area Model", COMPAQ Western Research Lab, 2001.
- [26] L. Spracklen and S. Abraham, "Chip Multithreading: Opportunities and Challenges," HPCA-11, Feb. 2005.
- [27] R. Stets, L.A. Barroso, et al., "A Detailed Comparison of TPC-C versus TPC-B," CAECW-3, January 2000.
- [28] S. Storino, A. Aipperspach, J. Borkenhagen, et al., "A commercial multithreaded RISC processor," ISSCC Dig. Tech. Papers, pages 234-235, Feb. 1998.
- [29] D. Tullsen, S. Eggers, and H. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," ISCA-22, June 1995, pages 392-403.
- [30] Standard Performance Evaluation Corporation, SPEC\*, <http://www.spec.org>, Warrenton, VA
- [31] Transaction Processing Performance Council, TPC-\*, <http://www.tpc.org>, San Francisco, CA
- [32] "XML Processing Performance in Java and .Net," [http://java.sun.com/performance/reference/whitepapers/XML\\_Test-1\\_0.pdf](http://java.sun.com/performance/reference/whitepapers/XML_Test-1_0.pdf)
- [33] Personal communication with Dr. Norm Jouppi